2-5

# SINC-LINK

# TIMEX-SINCLAIR USERS CLUB NEWSLETTER

## Toronto, Ontario

Trade Mark

PRINTED IN CANADA

B6146

## *LETTER FROM THE PRESIDENT*

*Welcome back from your summer holidays, fellow members. Our elections are now in the process of finding new executive officers which will be in the next Newsletter. There are several openings and anyone interested in serving on the executive, please leave your name with us.*

*I'm looking forward to a continued good year and since I'm stepping down as the President, all my best wishes to the new President of the Timex-Sinclair Users Club.*

*Greg Lloyd*
*(President)*

## EXECUTIVE OFFICERS

PRESIDENT: *Greg Lloyd*
SECRETARY: *George Chambers*
LIBRARIAN: *Martin Mauk*
TREASURER: *John Roach*
NEWS EDITOR: *Stan Piotrowski*
ACTIVITY DIRECTORS: *Brian Hammond, Ian Roberts*
LIASON OFFICER: *(Out-of-town members): Chris Hart*

## EDITOR'S NOTE

Again, I am asking for articles to be printed up in the Newsletter. As I've said before, it doesn't matter how trivial it may seem to you, pass it on to your Newsletter and have it printed. Remember, there are many new programmers out there who may have been struggling with the same problem you might have had and couldn't find the solution. By passing on these little bits of information, you may be helping another fellow club member. Also, you get to see your name in print and if anyone should ask, you can truthfully say that you have had some of your computer work published!!

*Stan Piotrowski*
*(News Editor)*

## BASIC PROGRAMMING

Well here we are again ready to do some more Basic Programming. In this issue, we will use some Hints & Techniques in order to enhance or make your Basic programming better.

Let's start off with the number of lines you are able to print onto your screen. The manual tells you that you can only print 22

lines (Ø to 21) yet it tells you that the display file or the screen size is 32 across by 24 lines down (Ø to 23). We actually do have access to this but there are 2 restrictions: we cannot print to 24 lines if there is an INPUT command or a SCROLL command. Otherwise, you can. The ZX-81 have several addresses in your RAM pack that it uses for itself and if you're not careful in POKEing these addresses, the computer can lock up or go into that wonderful display of a "blizzard in the middle of winter".

The address that keeps track of the number of lines on your screen is 16418. It normally contains "2" counting from the bottom line up which means that you cannot print on the bottom two lines. To print to the bottom 2 lines; POKE 16418,Ø. Now you can PRINT AT 23,Ø;"HELLO THERE"....BUT!!! you can't stop ther program there; it must continue on. If it stops, you know that you get error codes in the bottom left corner. All error codes are printed on line 23 so it would wipe out your PRINT statement. Therefore, if you enter something like the following lines:

```
1Ø CLS
2Ø POKE 16418,Ø
3Ø PRINT AT 1Ø,Ø;"ENTER A NUMBER"
4Ø IF INKEY$ = "" THEN GOTO 4Ø
5Ø LET A$=INKEY$
6Ø PRINT AT 23,1Ø;"YOUR NUMBER IS: ";A$
7Ø GOTO 3Ø
```

Now, you can add more printing data if you should want more on the screen. If you need to use a SCROLL or an INPUT, don't forget to POKE 16418,2 before these commands. Of course you can do the opposite which can be effective.

Enter the following program and you can see that the printing will stay on the bottom of the screen while you can scroll and input anything:

```
1Ø CLS
2Ø PRINT AT 21,Ø;"PRESS ANY KEY TO STOP"
3Ø POKE 16418,1Ø
4Ø FOR I=Ø TO 255
45 SCROLL
5Ø PRINT CHR$ I
6Ø IF INKEY$="" THEN GOTO 8Ø
7Ø GOTO 1ØØ
8Ø NEXT I
9Ø IF I=255 THEN GOTO 4Ø
1ØØ POKE 16418,2
11Ø FAST
12Ø CLS
13Ø SLOW
14Ø STOP
```

Have you ever noticed in some programs (especially game programs with machine code) that the first line is always "Ø". You know that if you entered a line number with a zero, then nothing would happen; the

ZX-81 refuses to accept it. Fear not; we can also change this too. Simply POKE 16510,0. Now list the program and your first line number will be a "0". Also notice that you cannot delete this line number. For programmers, they sometimes put in their copyright notice since most novice programmers cannot delete or edit it. By simply reversing the process, you can insert a line number that you can delete or edit; POKE 16510,1 or the same line number as the following line number in your program.

Whenever a program had the command PAUSE, did you notice how jerky it seems such as PAUSE 300. The reason is that it is "Bit" counting the frames on the TV screen. A much smoother effect for waiting a period of time is the FOR/NEXT loop such as:

```
100 FOR I = 1 TO 300
200 NEXT I
```

And finally, here's a short program that displays the calendar month for any year you choose. For those of you learning to program, go through line by line and try to determine exactly what and why the program is doing something. For example, in line #240, the numbers that represent some of the months is determined to have 31 days. It laso checks whether there was a leap year in the year you chose.

```
1   SLOW
2   CLS
3   PRINT AT 8,2;"1. HARDCOPY","2. SCREENCOPY"
4   IF INKEY$<>"1" AND INKEY$<>"2" THEN GOTO 4
5   LET P$="0"
6   IF INKEY$="1" THEN LET P$="1"
7   PRINT,,
10  PRINT "ENTER MONTH (1 TO 12)"
20  INPUT M
30  PRINT "ENTER YEAR"
40  INPUT Y
45  FAST
50  LET D=1
60  LET MON=M
70  LET YEAR=Y
80  DIM A$(12,9)
100 LET A$(1)="JANUARY  "
110 LET A$(2)="FEBRUARY "
120 LET A$(3)="MARCH    "
130 LET A$(4)="APRIL    "
140 LET A$(5)="MAY      "
150 LET A$(6)="JUNE     "
160 LET A$(7)="JULY     "
170 LET A$(8)="AUGUST   "
180 LET A$(9)="SEPTEMBER"
190 LET A$(10)="OCTOBER "
200 LET A$(11)="NOVEMBER "
210 LET A$(12)="DECEMBER "
220 DIM A(42)
230 LET END=30
```

```
240 IF M=1 OR M=3 OR M=5 OR M=7 OR M=8 OR M=10 OR M=12 THEN LET
END=31
250 IF M=2 THEN LET END =28
260 LET LEAP=0
270 IFY/4=INT (Y/4) AND Y/100<>INT (Y/100) THEN LET LEAP=1
280 IF Y/400=INT (Y/400) THEN LET LEAP=1
300 IF M=2 AND LEAP=1 THEN LET END=29
310 IF M=1 OR M=2 THEN LET M=M+12
320 IF M=13 OR M=14 THEN LET Y=Y-1
330 LET R=D+2*M+2+INT ((3*M+3)/5)+INT (Y/4)+Y-INT (Y/100)+INT
(Y/400)
340 LET NUM =(R/7-INT (R/7))*7
350 LET NUM =INT (NUM+ .5)
360 LET START = NUM
370 IF NUM=0 THEN LET START=7
380 LET DAY =1
390 FOR P=START TO 42
400 LET A(P)=DAY
410 LET DAY=DAY+1
420 NEXT P
430 CLS
440 PRINT TAB 10;A$(MON);" ";YEAR
450 PRINT,,"    SUN MON TUE WED THU FRI SAT"
470 FOR R=0TO5
480 FOR C=1+R*7 TO 7+R*7
490 LET CC=C-R*7
500 IF A(C)=0 THEN PRINT TAB (4*CC);" ";
510 IF A(C)=0 THEN GOTO 550
520 IF A(C) > END THEN PRINT TAB (4*CC);" ";
530 IF A(C) > END THAN GOTO 550
540 PRINT TAB (4*CC);A(C);
550 NEXT C
560 NEXT R
570 IF P$="1" THEN COPY
580 IF P$="1" THEN GOTO 1
590 SLOW
600 PRINT AT 21,0;"PRESS ANY KEY FOR ANOTHER MONTH"
610 IF INKEY$="" THEN GOTO 610
620 GOTO 1
```

## MACHINE CODE PROGRAMMING

On the last page you will find several piecs of information about the Z-80 chip which is the chip used in the ZX-81. They are all grouped in order that you may see the correlation between common M/C commands. There are few more things to learn before we can actually begin programming a game (or anything) in M/C.

As you learned in the last issue of our Newsletter, you can print any printable character to the screen by calling a special Print Routine within the Z-80 commands. This is fine for printing messages but it will not work for moving graphics around on the screen. There is another way and it, too, can be used to print messages but is more cumbersome and would take a great deal more patience and bytes.

Mr. Sinclair set up the ZX-81 so that 2 special addresses hold the location of the Display File (or D-File in the manual). In some computers, the display file is MEMORY MAPPED which means that always and forever, the same addresses hold the location of the screen positions. For example, in memory mapped computers, the display file may begin at location 2ØØØ onwards to a total of the number of print positions on the screen. If you wanted to print the letter "A" in the middle of the screen, then you would add the number of characters across (in the ZX-81 it is 32) times the number of rows down. In the ZX-81, the middle is about 1Ø lines down. Therefore, you would add 1Ø*32 or 32Ø to 2ØØØ to print the letter "A" in the middle. But not so with the ZX-81. As I stated above, the ZX-81 is NOT memory mapped but floats around in memory and since it is a "floater", it is a *"SYSTEM VARIABLE"*. If you check your manual on the System Variables, you will find that the addresses 16396 and 16397 hold the location of the beginning of the display file. If we PEEK these addresses at different times during programming, you will find that they are different each time. In actual fact, we don't really care where the D-File is located; all we want to do is print a character somewhere on the screen but in a place decided by us. (Which means doing some mathematics).

Oh yes, there is one more hitch in the floating D-File; at the beginning, there is always a character code of 118 and at the end of each line, there is a code of 118 which means that each line (except the first line) has actually 33 spaces instead of the normally 32 we have been accustomed to. We must take this into consideration each time we print something to the screen or check the character of what's on a particular screen location. Try this:

```
PRINT PEEK 16396 + 256 * PEEK 16397.
```

This will give you the address location of the start of D-File. If, for example, you PEEKed the above addresses and the result was 32ØØØ; PRINT PEEK 32ØØØ and you will get 118 which means the start of D-File.

Enter this short program to check how D-File works:

```
1Ø CLS
2Ø PRINT "A"
3Ø LET X=PEEK 16396 + 256 * PEEK 16397
4Ø LET X=X+1
5Ø PRINT X

RUN
```

Running through the program, the screen is cleared and the letter "A" is printed at the top left corner. In line 3Ø, "X" is the address of the beginning of the location of the display file. Since the first location is always code 118, in line 4Ø we add 1 to the address to get the first location of the printable part of the D-File. In line 5Ø, we PEEK this address contained in "X" which will be code 38. If you check in your manual, the code for 38 is the letter "A".

You should now see how this address holds the location of D-File and the fact that we let the computer look after the actual location. We don't really need to know where the D-File is located in memory but simply PEEK the variable addresses 16396 and 16397 to find it.

Now, if we can PEEK these addresses, we can also POKE a printable character into the D-File location. Don't forget, we don't POKE into the addresses 16396 & 16397 but only into their PEEK value.

Therefore, let's modify the above program slightly to POKE the letter "B" into the top left corner of our screen:

```
DELETE LINE 10
40 POKE X,39

RUN
```

The letter "B" will have appeared at the top left corner of the screen. The whole program above is in Basic so is rather slow but if it is done in machine language, it is done very quickly. In fact, it moves so fast that we will actually slow the routine down by about 10000 cycles in order that we can use the routine in a game!!! Notice, 10000 cycles. That's like using in Basic:

```
FOR I = 1 TO 10000
NEXT I
```

I showed you this routine before but now you should see the logic in it. First of all, we find the start of the loaction of D-File, then POKE a character code into it, add 1 to the address, go back and do it again. The big thing about this program is that it will crash and you have to unplug your computer to get out of it since it doesn't take into account all codes 118 and the end of D-File; in fact it will go on forever POKEing the codes into memory. All I'm trying to show you is how fast this routine is in comparison to the previous machine code printing routine. But, this routine is used the most in machine language programming. There is one thing more to remember and it is important: the register HL must be used whenever the above routine is being used and normally DE is used to hold the screen position. (e.g. the ZX-81 has 32 horizontal positions by 24 vertical positions or 768 possible positions that can be displayed on the screen.

Enter a REM line with 12 characters then enter the following codes:

```
10 REM 123456789012

42, 12,  64        :LD HL (16396)
17,  1,   0        :LD DE, 1
25                 :ADD HL,DE
54,  38            :LD (HL), 38
35                 :INC HL
24,  251           :JR -5

RAND USR 16514
```

Immediately the screen was filled with the letter "A" - so fast that you were unable to see the start or finish.

The last area at the present time before starting our "Space Game" is that of K-Scan or keyboard scan routine. We must be able to control our "Lasers" or the "Spaceship's" movements and in order to do this, we must be able to use the keyboard, obviously.

There are many subroutines in ROM that we make extensive use of which means we don't "REWRITE" the ROM routines and one of the routines is K-Scan. When we call this routine (in machine language: CALL 688), it returns with the value of the last key pressed in the HL register. If we Compare the values we are looking for with the values in the HL register, we can do something and if the values are not what we want, then do nothing. For example, if we wish to have the "Y" key to move our Spaceship up, we compare the value in the HL register pair to the value of the letter "Y".

Note: it is NOT the character value of "Y" but the keyboard section where the letter "Y" is contained. If you have a copy of Toni Baker's book "Mastering Machine Code On Your ZX-81" (an excellent book on machine code), beginning on page 88, she shows you what the sections are about. Basically, the keyboard is divided up into several sections (hardware and software).

The top row of key is one section; the next another section, etc. Then the sections are divided vertically as well beginning with 1, Q, A and SHIFT. This is a very brief explanation of it and if anyone has added an external keyboard, this was actually done in hardware. According to Toni Baker's book, we can use whole sections to move the spaceship. For example, the keys 1, 2, and 3 could be one type of movement. But if we wanted to get specific and have only the key labelled 1 to do something, then we must examine both the "H" and the "L" of the hl register pair. In the above sections, we need only to compare the "L" register.

I'll let you find out what the returned value is of each key pressed in the following Basic program. You can print them out for yourself or jot them down somewhere. Generally, you will only be concerned with 5 specific keys for most games: the left, right, up, and down movements and a fire-the-missile key.

```
10 SCROLL
20 IF INKEY$ = "" THEN GOTO 20
30 LET X = PEEK 16421
40 LET Y = PEEK 16422
50 LET A$ = INKEY$
60 PRINT A$;": LOBYTE=";X, "HIBYTE=";Y
70 GOTO 10
```

What happens in the above program is that the "LOBYTE" will be what is returned in the "L" register and the "HIBYTE" will be what is returned in the "H" register. In machine code, you would CALL K-SCAN and then compare the HL register with what your expected results would

be and then continue execution of the machine code program

Well that's it for this Newsletter for Machine Code programming. In the next issue, we will begin to actually program a Space Invaders game.

Since there are several new members in our club since we first began our Newsletters, I thought evry once in awhile I would print an article from some of our first Newsletters. Here then is an article by one of former members and very active participant (those of us that knew him):

**A SHORT PROGRAM IN FORTH (ZX FORTH)**
*by J.J. Castillos*

If you are like me, you got your copy of ZX FORTH, loaded it, played a bit with it and after realizing how different it was from Basic, you put it away for awhile. Just now I am beginning to grasp the elementary notions of FORTH programming and I would like to share with you a short program which shows some simple graphic and printing routines (each # represents one blank space). Compare the speed of execution with a similar BASIC program, or example, any one of the Basic games you may have which starts by drawing a black border aroubd the screen.

```
: G 24 1 DO CR ."##" 128 EMIT ."################################" 128
EMIT LOOP ; NEWLINE  !note: that's 30 spaces!
: A."##" 31 Ø DO 128 EMIT LOOP ; NEWLINE
: M 1Ø 1 DO CR LOOP ; NEWLINE
: E ."##" 128 EMIT ."####THIS IS A DEMONSTRATION" CR CR ; NEWLINE
: S ."##" 128 EMIT ."#####OF FORTH GRAPHICS..." CR CR ; NEWLINE
: X ."##" 128 EMIT ."################" ; NEWLINE !note: 14 spaces!
```

After making the above definitions exactly as listed, you can run the program by entering the following list of words:

TASK G A HOME A M E S X  Newline

and see what happens. Remember that after the laborious entering of the definitions, these stay in memory and are added to your Vocabulary. The actual program is the last short line of words.

Printed in Canada by Triune Printing 

The Z80 has eight directly accessible 8 bit registers, A,B,C, D,E,F,H and L, and four sixteen bit registers IX, IY, the stack pointer SP and the programme counter PC. The eight 8 bit registers are sometimes used as four sixteen bit registers AF,BC,DE,HL. A is the accumulator, F holds flags and HL is used to point to an address in memory.

In the following n is a single byte number, nn is a two byte number, d is a displacement and (nn) is the contents of memory location nn e.g (HL) is the contents of the address held in HL.

1 Load

Mnemonic LD. Example LD A,C- copy the contents of C into A.

1.1 8 bit register to register e.g LD A, C

The contents of any of the registers A,B,C,D,E,H,L may be copied one to another.

1.2 8 bit memory to register e.g LD A, (HL)

(HL), (IX+d) or (IY+d), may be copied to any of the registers A,B,C,D,E, H,L. (BC), (DE) or (nn) may be copied to A.

1.3 8 bit register to memory e.g LD (HL), A

A,B,C,D,E,H,L may be copied to (HL), (IX+d), (IY+d). A may be copied to (BC), (DE) or (nn).

1.4 8 bit register or memory immediate e.g LD A, n

A value n may be loaded into A,B,C,D,E,H,L, (HL), (IX+d), (IY+d).

1.5 16 bit register to register e.g LD SP, HL

The contents of HL, IX or IY may be copied to SP.

1.6 16 bit memory to register e.g LD BC, (nn)

(nn) may be copied to BC, DE, HL, IX, IY, SP.

1.7 16 bit register to memory e.g LD (nn), BC

BC, DE, HL, IX, IY, SP may be copied to (nn)

1.8 16 bit register immediate e.g LD BC, nn

nn may be loaded into BC, DE, HL, IX, IY, SP

2 Push and Pop e.g PUSH HL POP HL

A PUSH instruction copies the contents of a named 16 register to the bottom of the stack and decrements the stack pointer twice. A POP instruction does the reverse. AF,BC,DE,HL,IX,IY may be PUSHed or POPped.

3 Exchange e.g EX AF AF'

An exchange instruction swops the contents of the named pair of 16 bit registers or memory locations. In the case of EX and EXX the contents are swopped with an otherwise inaccessible set of duplicate registers. Exchanges can be made between HL and DE, HL and (SP), (SP) and IX,(SP) and IY, AF and AF', BCDEHL and BCDEHL'.

4 8 bit add and subtract e.g ADD A,E SBC A,D

A,B,C,D,E,H,L,(HL),n,(IX+d), (IY+d) may be added or subtracted to or from the accumulator with or without the carry flag.

5 8 bit AND OR and XOR e.g AND C

A,B,C,D,E,H,L,(HL),n, (IX+d), (IY+d) may be combined with the accumulator using any of the three logical operators above.

6 Compare e.g CP C

Compare is like subtract except the flags only and not the contents of the accumulator are affected. A,B,C,D,E,H,L,(HL),n (IX+d), (IY+d) may be compared with the accumulator.

7 8 bit increment and decrement e.g INC B

A,B,C,D,E,H,L,(HL), (IX+d), (IY+d) may be incremented or decremented.

8 16 bit increment and decrement e.g INC BC

BC,DE,HL,IX,IY,SP may be incremented or decremented.

9 16 bit add and subtract e.g ADD HL, BC

BC,DE,HL,IX may be added with or without carry or subtracted with carry only to or from HL. BC,DE,SP,IX may be added without carry to IX. BC,DE,SP, IY may be added without carry to IY.

10 Jump, call and return

The flag register, F, contains a carry flag, C, which is set on carry, a parity flag P, which is set if a result is even, a sign flag, S, which is set if a result is negative, an overflow flag, V, which is set on overflow, and a zero flag, Z, which is set on zero. These flags can be used to control jumps, subroutine calls and returns.

10.1 Jump e.g JP NC, nn

The following jumps to an address nn are possible: absolute jump (JP); jump on zero or not zero (JP Z and JP NZ); jump on carry or not carry (JP C and JP NC); jump on positive or negative (JP P and JP M); jump on P/V = 1 or P/V = 0 (JP PE and JP PO). The following relative jumps to an address d relative to the current position are available where d is interpreted as lying in the range - 128 to 127: absolute relative jump (JR); relative jump on zero or not zero (JR Z or JR NZ); relative jump on carry or not carry (JR C and JR NC). Jumps may be made to the addresses held in HL,IX or IY. (JP (HL), JP(IX), JP(IY)). The DJNZ instruction decrements the B register and jumps to d if B is non zero.

10.2 Call e.g Call Z nn

If the call condition is met the current contents of the programme counter is PUSHed on to the stack and a jump is made to address nn. The following calls may be made: absolute call (CALL); call on zero or not zero (CALL Z or CALL NZ); call on carry or not carry (CALL C or CALL NC); call on positive or negative (CALL P or CALL M); call on P/V = 1 or P/V = 0 (CALL PE or CALL PO).

10.3 Return e.g RET PO

If the return condition is met the value in the stack is POPped into the programme counter causing a jump back to the position following the previous call. Return conditions are available to match each call condition. Returns can also be made from the interrupt and the non-maskable interrupt (RETI and RETN)

11 Bit instructions

The eight bits in each register are numbered from 0 to 7 inclusive from right to left. Each of the following operations may be performed on the A,B,C,D, E,H,L registers and on (HL), (IX+d) and (IY+d).

11.1 Bit test e.g BIT 2, B

The bit test instruction sets the zero flag to the opposite of the named bit. Any bit may be tested.

11.2 Bit set e.g SET 5, D

Any bit may be set.

11.3 Bit reset e.g RES 7, H

Any bit may be reset.

11.4 Rotate left e.g RL B

Bit 7 is copied to the carry, the carry is copied to bit O and all other bits are copied one place to the left.

11.5 Rotate right e.g RR D

Bit O is copied to the carry, the carry is copied to bit 7 and all other bits are copied one place to the right.

11.6 Rotate left circular e.g RLC E

Bit 7 is copied to the carry and to bit O. All other bits are copied one place to the left.

11.7 Rotate right circular e.g RRC A

Bit O is copied to the carry and to bit 7. All other bits are copied one place to the right.

11.8 Shift left arithmetic e.g SLA (HL)

All bits are copied one place to the left, bit 7 is copied to the carry and bit O is reset.

11.9 Shift right arithmetic e.g SRA A

All bits are copied one place to the right, bit O is copied to the carry and bit 7 is copied to itself.

11.10 Shift right logical e.g SRL (IX+d)

As shift right arithmetic but with bit 7 reset.

11.11 Rotate left digit and rotate right digit e.g RLD

In a rotate left bits O to 3 of A are copied to bits O to 3 of (HL); bits O to 3 of (HL) are copied to bits 4 to 7 of (HL); bits 4 to 7 of (HL) are copied to bits O to 3 of A.

12 Accumulator operations

12.1 Complement accumulator CPL

Every set bit is reset, every reset bit is set.

12.2 Negate accumulator NEG

Complement and add one.

12.3 Complement or set carry CCF or SCF

CCF complements the carry; SCF sets the carry.

12.4 Decimal adjust DAA

Corrects the results of BCD addition and subtraction.

13 Restart e.g RST 20

Save the programme counter on the stack and jump to location 8*nH where nH is the number in hexa decimal.

14 Block handling e.g LDI

LDI: Move one byte from (DE) to (HL) and decrement BC. LDIR: As LDI but repeat until BC = 0.

LDD: Move one byte from (DE) to (HL) and decrement BC, DE and HL.

LDDR: As LDD but repeat until BC=0.

CPI: Compare A and (HL), increment HL and decrement BC. CPIR: As CPI but repeat until BC = 0. CPD: As CPI but decrement HL. CPDR: As CPD but repeat until BC = 0.

15 Flip - flop e.g DI

DI: reset interrupt flip - flop.

EI: set interrupt flip - flop.

# Z 80 OP Codes

| DEC | HEX | OP CODE |
|---|---|---|
| 0 | 00 | NOP |
| 1 | 01nn | LD BC,nn |
| 2 | 02 | LD (BC),A |
| 3 | 03 | INC BC |
| 4 | 04 | INC B |
| 5 | 05 | DEC B |
| 6 | 06n | LD B,n |
| 7 | 07 | RLCA |
| 8 | 08 | EX AF,AF' |
| 9 | 09 | ADD HL,BC |
| 10 | 0A | LD A,(BC) |
| 11 | 0B | DEC BC |
| 12 | 0C | INC C |
| 13 | 0D | DEC C |
| 14 | 0En | LD C,n |
| 15 | 0F | RRCA |
| 16 | 10d | DJNZ d |
| 17 | 11nn | LD DE,nn |
| 18 | 12 | LD (DE),A |
| 19 | 13 | INC DE |
| 20 | 14 | INC D |
| 21 | 15 | DEC D |
| 22 | 16n | LD D,n |
| 23 | 17 | RLA |
| 24 | 18d | JR d |
| 25 | 19 | ADD HL,DE |
| 26 | 1A | LD A,(DE) |
| 27 | 1B | DEC DE |
| 28 | 1C | INC E |
| 29 | 1D | DEC E |
| 30 | 1En | LD E,n |
| 31 | 1F | RRA |
| 32 | 20d | JR NZ,d |
| 33 | 21nn | LD HL,nn |
| 34 | 22nn | LD (nn),HL |
| 35 | 23 | INC HL |
| 36 | 24 | INC H |
| 37 | 25 | DEC H |
| 38 | 26n | LD H,n |
| 39 | 27 | DAA |
| 40 | 28d | JR Z,d |
| 41 | 29 | ADD HL,HL |
| 42 | 2Ann | LD HL,(nn) |
| 43 | 2B | DEC HL |
| 44 | 2C | INC L |
| 45 | 2D | DEC L |
| 46 | 2En | LD L,n |
| 47 | 2F | CPL |
| 48 | 30d | JR NC,d |
| 49 | 31nn | LD SP,nn |
| 50 | 32nn | LD (nn),A |
| 51 | 33 | INC SP |
| 52 | 34 | INC (HL) |
| 53 | 35 | DEC (HL) |
| 54 | 36n | LD (HL),n |
| 55 | 37 | SCF |
| 56 | 38d | JR C,d |
| 57 | 39 | ADD HL,SP |
| 58 | 3Ann | LD A,(nn) |
| 59 | 3B | DEC SP |
| 60 | 3C | INC A |
| 61 | 3D | DEC A |
| 62 | 3En | LD A,n |
| 63 | 3F | CCF |
| 64 | 40 | LD B,B |
| 65 | 41 | LD B,C |
| 66 | 42 | LD B,D |
| 67 | 43 | LD B,E |
| 68 | 44 | LD B,H |
| 69 | 45 | LD B,L |
| 70 | 46 | LD B,(HL) |
| 71 | 47 | LD B,A |
| 72 | 48 | LD C,B |
| 80 | 50 | LD D,B |
| 88 | 58 | LD E,B |
| 96 | 60 | LD H,B |
| 104 | 68 | LD L,B |
| 112 | 70 | LD (HL),B |
| 120 | 78 | LD A,B |
| 128 | 80 | ADD A,B |
| 136 | 88 | ADC A,B |
| 144 | 90 | SUB B |
| 152 | 98 | SBC A,B |
| 160 | A0 | AND B |
| 168 | A8 | XOR B |
| 176 | B0 | OR B |
| 184 | B8 | CP B |
| 192 | C0 | RET NZ |
| 193 | C1 | POP BC |
| 194 | C2nn | JP NZ,nn |
| 195 | C3nn | JP nn |
| 196 | C4nn | CALL NZ,nn |
| 197 | C5 | PUSH BC |
| 198 | C6n | ADD A,n |
| 199 | C7 | RST 0H |
| 200 | C8 | RET Z |
| 201 | C9 | RET |
| 202 | CAnn | JP Z,nn |
| 203 | CB | see below |
| 204 | CCnn | CALL Z,nn |
| 205 | CDnn | CALL nn |
| 206 | CEn | ADC A,n |
| 207 | CF | RST 8H |
| 208 | D0 | RET NC |
| 209 | D1 | POP DE |
| 210 | D2nn | JP NC,nn |
| 211 | D3 | OUT A |
| 212 | D4nn | CALL NC,nn |
| 213 | D5 | PUSH DE |
| 214 | D6n | SUB n |
| 215 | D7 | RST 10H |
| 216 | D8 | RET C |
| 217 | D9 | EXX |
| 218 | DAnn | JP C,nn |
| 219 | DB | IN A |
| 220 | DCnn | CALL C,nn |
| 221 | DD | see below |
| 222 | DEn | SBC A,n |
| 223 | DF | RST 18H |
| 224 | E0 | RET PO |
| 225 | E1 | POP HL |
| 226 | E2nn | JP PO,nn |
| 227 | E3 | EX (SP),HL |
| 228 | E4nn | CALL PO,nn |
| 229 | E5 | PUSH HL |
| 230 | E6n | AND n |
| 231 | E7 | RST 20H |
| 232 | E8 | RET PE |
| 233 | E9 | JP (HL) |
| 234 | EAnn | JP PE,nn |
| 235 | EB | EX DE,HL |
| 236 | ECnn | CALL PE,nn |
| 237 | ED | see below |
| 238 | EEn | XOR n |
| 239 | EF | RST 28H |
| 240 | F0 | RET P |
| 241 | F1 | POP AF |
| 242 | F2nn | JP P,nn |
| 243 | F3 | DI |
| 244 | F4nn | CALL P,nn |
| 245 | F5 | PUSH AF |
| 246 | F6n | OR n |
| 247 | F7 | RST 30H |
| 248 | F8 | RET M |
| 249 | F9 | LD SP,HL |
| 250 | FAnn | JP M,nn |
| 251 | FB | EI |
| 252 | FCnn | CALL M,nn |
| 253 | FD | see below |
| 254 | FEn | CP n |
| 255 | FF | RST 38H |

## Two byte instructions

Each of the following codes is preceded by CB (decimal 203)

| DEC | HEX | OP CODE |
|---|---|---|
| 0 | 00 | RLC B |
| 1 | 01 | RLC C |
| 2 | 02 | RLC D |
| 3 | 03 | RLC E |
| 4 | 04 | RLC H |
| 5 | 05 | RLC L |
| 6 | 06 | RLC (HL) |
| 7 | 07 | RLC A |
| 8 | 08 | RRC B |
| 16 | 10 | RL B |
| 24 | 18 | RR B |
| 32 | 20 | SLA B |
| 40 | 28 | SRA B |
| 56 | 38 | SRL B |
| 64 | 40 | BIT 0,B |
| 72 | 48 | BIT 1,B |
| 80 | 50 | BIT 2,B |
| 88 | 58 | BIT 3,B |
| 96 | 60 | BIT 4,B |
| 104 | 68 | BIT 5,B |
| 112 | 70 | BIT 6,B |
| 120 | 78 | BIT 7,B |
| 128 | 80 | RES 0,B |
| 136 | 88 | RES 1,B |
| 144 | 90 | RES 2,B |
| 152 | 98 | RES 3,B |
| 160 | A0 | RES 4,B |
| 168 | A8 | RES 5,B |
| 176 | B0 | RES 6,B |
| 184 | B8 | RES 7,B |
| 192 | C0 | SET 0,B |
| 200 | C8 | SET 1,B |
| 208 | D0 | SET 2,B |
| 216 | D8 | SET 3,B |
| 224 | E0 | SET 4,B |
| 232 | E8 | SET 5,B |
| 240 | F0 | SET 6,B |
| 248 | F8 | SET 7,B |

Op codes 40 to BF (decimal 64 to 191) and CB 00 to CB FF (decimal 203 0 to 203 255) fall naturally into groups of eight. Successive Op codes in each group use the next in the sequence B,C,D,E,H,L, (HL),A as the source or operand as appropriate. For example Op codes of the form n3 and nB where n=4,5,6,7,A or B all refer to E.

Op codes 40 to 47 and CB 00 to CB 07 are listed in their entirety but subsequently the first of each group only is shown.

Each of the following codes is preceded by ED (decimal 237).

| DEC | HEX | OP CODE |
|---|---|---|
| 66 | 42 | SBC HL,BC |
| 67 | 43nn | LD (nn),BC |
| 68 | 44 | NEG |
| 69 | 45 | RETN |
| 71 | 47 | LD I,A |
| 74 | 4A | ADC HL,BC |
| 75 | 4Bnn | LD BC,(nn) |
| 77 | 4D | RETI |
| 79 | 4F | LD R,A |
| 82 | 52 | SBC HL,DE |
| 83 | 53nn | LD (nn),DE |
| 87 | 57 | LD A,I |
| 90 | 5A | ADC HL,DE |
| 91 | 5Bnn | LD DE,(nn) |
| 95 | 5F | LD A,R |
| 98 | 62 | SBC HL,HL |
| 103 | 67 | RRD |
| 106 | 6A | ADC HL,HL |
| 111 | 6F | RLD |
| 114 | 72 | SBC HL,SP |
| 115 | 73nn | LD (nn),SP |
| 122 | 7A | ADC HL,SP |
| 123 | 7Bnn | LD SP,(nn) |
| 160 | A0 | LDI |
| 161 | A1 | CPI |
| 168 | A8 | LDD |
| 169 | A9 | CPD |
| 176 | B0 | LDIR |
| 177 | B1 | CPIR |
| 184 | B8 | LDDR |
| 185 | B9 | CPDR |

Each of the following codes is preceded by DD (decimal 221) when referring to the IX register or FD (decimal 253) when referring to the IY register. For clarity IX only is used below.

| DEC | HEX | OP CODE |
|---|---|---|
| 9 | 09 | ADD IX,BC |
| 25 | 19 | ADD IX,DE |
| 33 | 21nn | LD IX,nn |
| 34 | 22nn | LD (nn),IX |
| 35 | 23 | INC IX |
| 41 | 29 | ADD IX,IX |
| 42 | 2Ann | LD IX,(nn) |
| 43 | 2B | DEC IX |
| 52 | 34d | INC (IX+d) |
| 53 | 35d | DEC (IX+d) |
| 54 | 36dn | LD (IX+d),n |
| 57 | 39 | ADD IX,SP |
| 70 | 46d | LD B,(IX+d) |
| 78 | 4Ed | LD C,(IX+d) |
| 86 | 56d | LD D,(IX+d) |
| 94 | 5Ed | LD E,(IX+d) |
| 102 | 66d | LD H,(IX+d) |
| 110 | 6Ed | LD L,(IX+d) |
| 112 | 70d | LD (IX+d),B |
| 113 | 71d | LD (IX+d),C |
| 114 | 72d | LD (IX+d),D |
| 115 | 73d | LD (IX+d),E |
| 116 | 74d | LD (IX+d),H |
| 117 | 75d | LD (IX+d),L |
| 119 | 77d | LD (IX+d),A |
| 126 | 7Ed | LD A,(IX+d) |
| 134 | 86d | ADD A,IX+d) |
| 142 | 8Ed | ADC A,(IX+d) |
| 150 | 96d | SUB (IX+d) |
| 158 | 9Ed | SBC A,(IX+d) |
| 166 | A6d | AND (IX+d) |
| 174 | AEd | XOR (IX+d) |
| 182 | B6d | OR (IX+d) |
| 190 | BEd | CP (IX+d) |
| 225 | E1 | POP IX |
| 227 | E3 | EX (SP),IX |
| 229 | E5 | PUSH IX |
| 233 | E9 | JP (IX) |
| 249 | F9 | LD SP,IX |

## Three byte instructions

Each of the following codes is preceded by DD CB (decimal 221 203) when referring to the IX register or FD CB (decimal 253 203) when referring to the IY register. For clarity IX only is used below.

| DEC | HEX | OP CODE |
|---|---|---|
| 6 | d06 | RLC (IX+d) |
| 14 | d0E | RRC (IX+d) |
| 22 | d16 | RL (IX+d) |
| 30 | d1E | RR (IX+d) |
| 38 | d26 | SLA (IX+d) |
| 46 | d2E | SRA (IX+d) |
| 62 | d3E | SRLR (IX+d) |
| 70 | d46 | BIT 0,(IX+d) |
| 78 | d4E | BIT 1,(IX+d) |
| 86 | d56 | BIT 2,(IX+d) |
| 94 | d5E | BIT 3,(IX+d) |
| 102 | d66 | BIT 4,(IX+d) |
| 110 | d6E | BIT 5,(IX+d) |
| 118 | d76 | BIT 6,(IX+d) |
| 126 | d7E | BIT 7,(IX+d) |
| 134 | d86 | RES 0,(IX+d) |
| 142 | d8E | RES 1,(IX+d) |
| 150 | d96 | RES 2,(IX+d) |
| 158 | d9E | RES 3,(IX+d) |
| 166 | dA6 | RES 4,(IX+d) |
| 174 | dAE | RES 5,(IX+d) |
| 182 | dB6 | RES 6,(IX+d) |
| 190 | dBE | RES 7,(IX+d) |
| 198 | dC6 | SET 0,(IX+d) |
| 206 | dCE | SET 1,(IX+d) |
| 214 | dD6 | SET 2,(IX+d) |
| 222 | dDE | SET 3,(IX+d) |
| 230 | dE6 | SET 4,(IX+d) |
| 238 | dEE | SET 5,(IX+d) |
| 246 | dF6 | SET 6,(IX+d) |
| 254 | dFE | SET 7,(IX+d) |